UVOS CLIENT MANUAL

UNICORE Team

Document Version:	1.6.0
Component Version:	1.7.0
Date:	14 01 2013

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.
This work was co-funded by the EC Chemomentum project under the FP6 Grant Agreement Nr. IST-033437.

Contents

1 Introduction **1**

2 Installation **1**

2.1 Installation from RPM package (RedHat distributions) 1

2.2 Installation from the DEB package (Debian distributions) 2

2.3 Installation from the archive 2

3 Using the command line UVOS client **2**

4 Using other available clients **3**

5 Developing with uvos-client **3**

6 Client configuration **3**

6.1 Configuring PKI trust settings 4

6.2 Configuring the credential 9

7 Commands reference **12**

UNICORE VO Service (UVOS) is a client-server system, developed to be used as an additional tool for large distributed systems, providing a solution for grid users management. Grid systems, especially UNICORE grid middleware, are the mainspring of the UVOS system. UVOS can be used with different systems, however is designed primarily to support UNICORE grid middleware.

For more information about UVOS visit http://uvos.chemomentum.org.

1 Introduction

This package called *uvos-client* contains client side libraries and a command line application (CLC) for accessing UVOS Server.

For a general introduction of UVOS please refer to its website: http://uvos.chemomentum.org

A complete overview of the UVOS is available in the initial chapters of the "UVOS Server manual" available there.

2 Installation

UVOS Client is distributed in the following formats:

1. As a platform independent archive.
2. As a binary, platform-specific package, available currently for Scientific Linux 5, Scientific Linux 6 and Debian 6 platforms. The packages are tested on the enumerated platforms, but should work without any problems with other versions of similar distributions (e.g. version for SL6 works well on Centos 6 or recent Fedora distributions. Differences between SL5 and SL6 version are only in the RPM tools used to create packages (so SL5 version should be more universal, while SL6 version can require a newer rpm software).

Depending on the installation source used, installation method and paths after installation are different. In case of platform-specific packages, the default configuration of the client is placed in `/etc/unicore/uvos-clc/uvosClient.conf` Upon a first start of the client it will be copied to `.uvos-clc/` folder in your home directory. You should edit it there (see Configuration Section 6 for details). The executable program is called `uvos-clc`.

2.1 Installation from RPM package (RedHat distributions)

The preferred way is to use Yum to install (and subsequently update) UVOS Client.

To perform the Yum installation, EMI Yum repository must be installed first. Refer to the EMI release documentation (available at the EMI website http://www.eu-emi.eu/releases) for

detailed instructions. Typically installation of the EMI repository requires to download a single RPM file and install it.

After the EMI repository is configured, the following command installs UVOS Client:

```
$> yum install unicore-uvos-clc
```

2.2 Installation from the DEB package (Debian distributions)

The preferred installation way is to use apt to install and subsequently update UVOS Client.

To perform the apt installation, EMI apt repository must be installed first. Refer to the EMI release documentation (available at the EMI website http://www.eu-emi.eu/releases) for detailed instructions. Typically installation of the EMI repository requires to download a single DEB file and install it.

After the EMI repository is configured, the following command installs UVOS Client:

```
$> apt-get install unicore-uvos-clc
```

2.3 Installation from the archive

If installing using a portable archive:

1. Download the uvos-client archive from the UNICORE download site and unpack it.
2. The default configuration of the client is placed in `conf/uvosClient.conf` You should edit it there (see Configuration Section 6 for details). The program can be found in the `bin/` subdirectory and is called `uvoscmd.sh`

3 Using the command line UVOS client

UVOS Command Line Client can operate in batch or interactive mode. Invoking it without arguments provides you with usage instructions. When invoking UVOS CLC user must select one of the available commands and provide arguments for it.

UVOS CLC offers a built in help system: it allows you to list all commands and get help on each of them. The following command outputs help for the operation `addIdentity` (assuming you are invoked UVOS CLC in interactive mode, similarly you can invoke it in batch mode):

```
help addIdentity
```

Note that in interactive mode in order to pass arguments which contain spaces you should surround them with double quotes "like this".

The command line client uses a configuration file to get information about:

- UVOS server address.
- How to authenticate to the server.
- Trusted certificates for TLS connection.

See Configuration Section 6 for details.

4 Using other available clients

Other clients which are available in this package are usually not useful for UVOS users. The sole exception is SAMLSelfClient. This program gets all attributes which are defined for the identity making the call (i.e. this one which is set in client's configuration file).

The *WebClient* program serves as an example for developers only so you can ignore it.

5 Developing with uvos-client

If you intend to use this package as a library please refer to the JavaDocs documentation (it is available from the download site, and from the documentation site which is preferred, the most accurate source). Remember that there are two VO APIs implemented by this library:

- **SAML API** It uses SAML 2.0 and so is standard-complaint, can be used to access any version of UVOS server and possibly other SAML services. However it offers only a read-only access. If you can then use use this API. It is included in the package: pl.edu.icm.unicore.uvos.wsclient.samlapi
- **UVOS API** It uses an own UVOS web services interface. It is not guaranteed that using this API will work perfectly with older UVOS server releases (however most of fundamental functions should work). Also this API in not standards complaint. On the other hand it allows you to use all UVOS features. It is included in the package: pl.edu.icm.unicore.uvos.wsclient.api

Also note that if intend to simply add UVOS support for your XFire or WSRFLite or UAS service then you should use xfire-voutils or uas-authz packages which provides a ready to use solution for such cases.

For additional information please contact UVOS or UNICORE developers at: unicore-devel@lists.sourceforge.net

6 Client configuration

The client uses the configuration file to get information about:

- UVOS server address,

- clients identity which is used to authenticate the client to the UVOS server,
- trusted certificates for TLS connection.

The file location can be chosen via *-c file* argument or by setting the UVOSCLC_CONFIG shell variable. The Java Properties format is used. See:

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream)

for a formal discussion, however it is enough to know that:

- comments are started with #
- on each line (non empty and not commented out) there is one property and its value defined...
- ... with the syntax: propertyName=property value

UVOS can be contacted using insecure HTTP or secure HTTPS. Except closed test environments it is always strongly suggested to use secure HTTPS.

Clients can authenticate to the UVOS using two mechanisms:

- using X.509 certificate - it is taken from SSL session so to use this authentication you must use *https* protocol.
- using email and password - then any protocol can be used.

Below the valid properties are presented along with descriptions.

6.1 Configuring PKI trust settings

Public Key Infrastructure (PKI) trust settings are used to validate certificates. This is performed, in the first place when a connection with a remote peer is initiated over the network, using the SSL (or TLS) protocol. Additionally certificate validation can happen in few other situations, e.g. when checking digital signatures of various sensitive pieces of data.

Certificates validation is primarily configured using a set of initially trusted certificates of so called Certificate Authorities (CAs). Those trusted certificates are also known as *trust anchors* and their collection is called a *trust store*.

Except of *trust anchors* validation mechanism can use additional input for checking if a certificate being checked was not revoked and if its subject is in a permitted namespace.

UNICORE allows for different types of trust stores. All of them are configured using a set of properties.

- *Keystore trust store* - the only format supported in older UNICORE versions. Trusted certificates are stored in a single binary file in JKS or PKCS12 format. The file can be only manipulated using a special tool like JDK *keytool* or openssl (in case of PKCS12 format). This format is great if trust store should be in a single file or when compatibility with other Java solutions or older UNICORE releases is desired.

- *OpenSSL trust store* - allows to use a directory with CA certificates stored in PEM format, under precisely defined names: the CA certificates, CRLs, signing policy files and namespaces files are named <hash>.0, <hash>.r0, <hash>.signing_policy and <hash>.namespaces. Hash is the old hash of the trusted CA certificate subject name (in Openssl version > 1.0.0 use -suject_hash_old switch to generate it). If multiple certificates have the same hash then the default zero number must be increased. This format is the same as used by other then UNICORE popular middlewares as Globus and gLite. It is suggested when a common trust store with such middlewares is needed.
- *Directory trust store* - the most flexible and convenient option, suggested for all remaining cases. It allows to use a list of wildcard expressions, concrete paths of files or even URLs to remote files as a set of trusted CAs and in the same way for the CRLs. With this trust store administrator can simply configure all files (or all with a specified extension) in a directory to be used as a trusted certificates.

In all cases trust stores can be (and by default are) configured to be automatically refreshed.

Property name	**Type**	**Default value / mandatory**	**Description**
`uvos.truststore.-allowProxy`	[ALLOW, DENY]	`ALLOW`	Controls whether proxy certificates are supported.
`uvos.truststore.-type`	[keystore, openssl, directory]	*mandatory to be set*	The truststore type.
`uvos.truststore.-updateInterval`	integer number	`600`	How often the truststore should be reloaded, in seconds. Set to negative value to disable refreshing at runtime. *(runtime updateable)*
--- Directory type settings ---			
`uvos.truststore.-directoryConnect-ionTimeout`	integer number	`15`	Connection timeout for fetching the remote CA certificates in seconds.
`uvos.truststore.-directoryDiskCac-hePath`	filesystem path	-	Directory where CA certificates should be cached, after downloading them from a remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it.

Property name	Type	Default value / mandatory	Description
uvos.truststore.directoryEncoding	[PEM, DER]	PEM	For directory truststore controls whether certificates are encoded in PEM or DER.
uvos.truststore.directoryLocations.*	list of properties with a common prefix	-	List of CA certificates locations. Can contain URLs, local files and wildcard expressions. *(runtime updateable)*
--- Keystore type settings ---			
uvos.truststore.keystoreFormat	string	-	The keystore type (jks, pkcs12) in case of truststore of keystore type.
uvos.truststore.keystorePassword	string	-	The password of the keystore type truststore.
uvos.truststore.keystorePath	string	-	The keystore path in case of truststore of keystore type.
--- Openssl type settings ---			
uvos.truststore.opensslNsMode	[GLOBUS_EUGRIDPMA_EUGRIDPMA_GLOBUS, GLOBUS, EUGRIDPMA, GLOBUS_EUGRIDPMA_REQUIRE, EUGRIDPMA_GLOBUS_REQUIRE, GLOBUS_REQUIRE, EUGRIDPMA_REQUIRE, EUGRIDPMA_AND_GLOBUS, EUGRIDPMA_AND_GLOBUS_REQUIRE, IGNORE]	EUGRIDPMA_GLOBUS	In case of openssl truststore, controls which (and in which order) namespace checking rules should be applied. The *REQUIRE* settings will cause that all configured namespace definitions files must be present for each trusted CA certificate (otherwise checking will fail). The *AND* settings will cause to check both existing namespace files. Otherwise the first found is checked (in the order defined by the property).
uvos.truststore.opensslPath	filesystem path	/etc/grid-security/certificates	Directory to be used for opeenssl truststore.
--- Revocation settings ---			

Property name	Type	Default value / mandatory	Description
`uvos.truststore.crlConnectionTimeout`	integer number	`15`	Connection timeout for fetching the remote CRLs in seconds (not used for Openssl truststores).
`uvos.truststore.crlDiskCachePath`	filesystem path	-	Directory where CRLs should be cached, after downloading them from remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it. Not used for Openssl truststores.
`uvos.truststore.crlLocations.*`	list of properties with a common prefix	-	List of CRLs locations. Can contain URLs, local files and wildcard expressions. Not used for Openssl truststores. *(runtime updateable)*
`uvos.truststore.crlMode`	[REQUIRE, IF_VALID, IGNORE]	`IF_VALID`	General CRL handling mode. The IF_VALID setting turns on CRL checking only in case the CRL is present.
`uvos.truststore.crlUpdateInterval`	integer number	`600`	How often CRLs should be updated, in seconds. Set to negative value to disable refreshing at runtime. *(runtime updateable)*
`uvos.truststore.ocspCacheTtl`	integer number	`3600`	For how long the OCSP responses should be locally cached in seconds (this is a maximum value, responses won't be cached after expiration)
`uvos.truststore.ocspDiskCache`	filesystem path	-	If this property is defined then OCSP responses will be cached on disk in the defined folder.

Property name	Type	Default value / mandatory	Description
uvos.truststore.ocspLocalResponders.<NUMBER>	list of properties with a common prefix	-	Optional list of local OCSP responders
uvos.truststore.ocspMode	[REQUIRE, IF_AVAILABLE, IGNORE]	IF_AVAILABLE	General OCSP ckecking mode. REQUIRE should not be used unless it is guaranteed that for all certificates an OCSP responder is defined.
uvos.truststore.ocspTimeout	integer number	10000	Timeout for OCSP connections in miliseconds.
uvos.truststore.revocationOrder	[CRL_OCSP, OCSP_CRL]	OCSP_CRL	Controls overal revocation sources order
uvos.truststore.revocationUseAll	[true, false]	false	Controls whether all defined revocation sources should be always checked, even if the first one already confirmed that a checked certificate is not revoked.

Examples

Note
Various UNICORE modules use different property prefixes. Here we don't put any, but in practice you have to use the prefix (see the reference table above for the actual prefix). Also properties might need to be provided using different syntax, as XML.

Directory trust store, with a minimal set of options:

```
truststore.type=directory
truststore.directoryLocations.1=/trust/dir/*.pem
truststore.crlLocations=/trust/dir/*.crl
```

Directory trust store, with a complete set of options:

```
truststore.type=directory
truststore.allowProxy=DENY
truststore.updateInterval=1234
truststore.directoryLocations.1=/trust/dir/*.pem
```

```
truststore.directoryLocations.2=http://caserver/ca.pem
truststore.directoryEncoding=PEM
truststore.directoryConnectionTimeout=100
truststore.directoryDiskCachePath=/tmp
truststore.crlLocations.1=/trust/dir/*.crl
truststore.crlLocations.2=http://caserver/crl.pem
truststore.crlUpdateInterval=400
truststore.crlMode=REQUIRE
truststore.crlConnectionTimeout=200
truststore.crlDiskCachePath=/tmp
```

Openssl trust store:

```
truststore.type=openssl
truststore.opensslPath=/truststores/openssl
truststore.opensslNsMode=EUGRIDPMA_GLOBUS_REQUIRE
truststore.allowProxy=ALLOW
truststore.updateInterval=1234
truststore.crlMode=IF_VALID
```

Java keystore used as a trust store:

```
truststore.type=keystore
truststore.keystorePath=src/test/resources/certs/truststore.jks
truststore.keystoreFormat=JKS
truststore.keystorePassword=xxxxxx
```

6.2 Configuring the credential

UNICORE uses private key and a corresponding certificate (called together as a *credential*) to identify users and servers. Credentials might be provided in several formats:

- Credential can be obtained from a *keystore file*, encoded in JKS or PKCS12 format.
- Credential can be loaded as a pair of PEM files (one with private key and another with certificate),
- or from a pair of DER files,
- or even from a single file, with PEM-encoded certificates and private key (in any order).

The following table list all parameters which allows for configuring the credential. Note that nearly all options are optional. If not defined, the format is tried to be guessed. However some credential formats require additional settings. For instance if using *der* format the *keyPath* is mandatory as you need two DER files: one with certificate and one with the key (and the latter can not be guessed).

Property name	**Type**	**Default value / mandatory**	**Description**
`uvos.credential.-path`	filesystem path	*mandatory to be set*	Credential location. In case of *jks*, *pkcs12* and *pem* store it is the only location required. In case when credential is provided in two files, it is the certificate file path.
`uvos.credential.-format`	[jks, pkcs12, der, pem]	-	Format of the credential. It is guessed when not given. Note that *pem* might be either a PEM keystore with certificates and keys (in PEM format) or a pair of PEM files (one with certificate and second with private key).
`uvos.credential.-password`	string	-	Password required to load the credential.
`uvos.credential.-keyPath`	string	-	Location of the private key if stored separately from the main credential (applicable for *pem* and *der* types only),
`uvos.credential.-keyPassword`	string	-	Private key password, which might be needed only for *jks* or *pkcs12*, if key is encrypted with different password then the main credential password.
`uvos.credential.-keyAlias`	string	-	Keystore alias of the key entry to be used. Can be ignored if the keystore contains only one key entry. Only applicable for *jks* and *pkcs12*.

6.2.1 Examples

Note
Various UNICORE modules use different property prefixes. Here we don't put any, but in practice you have to use the prefix (see the reference table above for the actual prefix). Also properties might need to be provided using different syntax, as XML.

Credential as a pair of DER files:

```
credential.format=der
credential.password=the\!njs
credential.path=/etc/credentials/cert-1.der
credential.keyPath=/etc/credentials/pk-1.der
```

Credential as a JKS file (credential type can be autodetected in almost every case):

```
credential.path=/etc/credentials/server1.jks
credential.password=xxxxxx
```

6.2.2 Example configuration file

```
########################################
# Client general settings
########################################
uvos.port=51774
uvos.host=localhost

########################################
# SSL settings (for https only)
########################################
# do we authenticate ourself to the server?
client.sslAuthnEnabled=false

credential.format=jks
credential.path=/opt/keystore.jks
credential.keyAlias=mykey
credential.password=the!client

truststore.type=keystore
truststore.keystorePath=/opt/truststore.jks
truststore.keystorePassword=the!client
truststore.keystoreFormat=JKS

########################################
```

```
# AUTHN settings (for https only)
########################################
# do we authenticate with HTTP Basic authN (i.e. username & passwd) ↩
    ?
client.httpAuthnEnabled=true
client.httpUser=voadmin@localhost
client.httpPassword=
```

7 Commands reference

This section provides reference documentation about all commands which are available in UVOS Command Line Client. This reference can be generated by the command *helpAll*. Note that information generated by help system of the Command Line Client is always up to date.

```
Command: addEquivalentIdentity

Creates a new identity, which is alias of (is equivalent to)  ↩
    another, existing identity. Syntax:
  addEquivalentIdentity <dn|x509|email> <newIdentity> <dn|x509| ↩
      email> <equivalent>
The first argument is an identity type. Next is value, which is a  ↩
    simple string in case of 'dn' or
'email' types and a file name with X509 certificate in case of ' ↩
    x509' type.
It is the value of added identity. Next come parameters of  ↩
    eqivalent identity.
Note that label of the new identity will be the same as of it's  ↩
    eqivalent.

Example:
  addEquivalentIdentity email john@example.com email john@example. ↩
      org


~~~~~~~~~~~~~~
Command: addGroup

Adds a new group. Syntax:
  addGroup <parentGroupPath> <newGroupName>
Separate the parent group path elements with '/'. Example:
  addGroup /parentGroup newGroup

~~~~~~~~~~~~~~
Command: addIdentity

Creates a new identity. Syntax:
  addIdentity <dn|x509|email> <value> [label]
```

```
or:  addIdentity email <value> passwd <password> [label]
The first argument is an identity type. Next is value, which is a  ↩
    simple string in case of 'dn' or 'email'
types and a file name with certificate in case of 'x509' type. 3rd  ↩
    (optional) argument is a friendly
(unique) name for the identity. It is common for all equivalent  ↩
    identities. When adding email type identity
it is also possible to set it's password.

Example:
  addIdentity email test@example.com Johnny


~~~~~~~~~~~~~~
Command: addNotification

Adds a new notification definition for the given action. Optionally ↩
     you can register it only in the context of a group
Syntax:
   addNotification <action> <recipents> [groupFilter]
Example:
   addNotification addGroup receiver@example.com /Math-VO

~~~~~~~~~~~~~~
Command: addToGroup

Adds an identity to a group. Syntax:
  addToGroup <dn|x509|email> <identity> <groupPath>
The first argument is an identity type. Next is value, which is a  ↩
    simple string in case of 'dn' or 'email'
types and a file name with X509 certificate in case of 'x509' type. ↩
     It is the value of added identity.
The last argument is the group path.

Example:
  addToGroup email john@example.com /group/subgroup


~~~~~~~~~~~~~~
Command: areEquivalent

Checks if two given identities are equivalent.
Syntax:
   areEquivalent <dn|x509|email> <identity1> <dn|x509|email> < ↩
       identity2>
Example:
  areEquivalent email ann@example.com email ann@example.org


~~~~~~~~~~~~~~
```

```
Command: changeLabel

Changes unique name of identity, which is common for all equivalent ↩
     identities. Syntax:
  changeLabel <dn|x509|email> <identity> <newLabel>
Example:
  changeLabel email ann@example.com Ann


~~~~~~~~~~~~~~
Command: changePasswd

Changes the given identity password (or deletes it).
Syntax:
   changePasswd email <identity> [newPasswd]
If new password is not given then the password is removed.Note that ↩
     it is technically possible to set also
password for other types of identity but it makes no sense.Example:
   changePasswd email ann@example.com SecreT


~~~~~~~~~~~~~~
Command: copyGroup

Copies or moves a group to a different parent group. Syntax:
  copyGroup <grouptoBeCopiedPath> <newParentGroupPath> <doMove> [ ↩
      newName]
Example:
  copyGroup /group/subgroup /parent true movedG
will create group /parent/movedG with the same contents as /group/ ↩
    subgoup has, then /group/subgroup will be
deleted.

~~~~~~~~~~~~~~
Command: disableAttribute

Disables an attribute. Syntax:
  disableAttribute <global> <dn|x509|email> <identity> < ↩
      AttributeName> [value]
  disableAttribute <group> <groupPath> <AttributeName> [value]
  disableAttribute <ig> <dn|x509|email> <identity> <groupPath>< ↩
      AttributeName> [value]
The first argument specifies what kind of attribute will be changed ↩
    : group attribute, identity global
attribute, or identity attribute valid only in particular group. If ↩
     the optional 'value' argument is set
then only this value of attribute will be disabled.
Example:
  disableAttribute ig email ann@example.com /group urn:unicore: ↩
      attrType:user:proffession scientist
```

```
~~~~~~~~~~~~~~
Command: enableAttribute

Enables an attribute. Syntax:
  enableAttribute <global> <dn|x509|email> <identity> < ↩
      AttributeName> [value]
  enableAttribute <group> <groupPath> <AttributeName> [value]
  enableAttribute <ig> <dn|x509|email> <identity> <groupPath>< ↩
      AttributeName> [value]
The first argument specifies what kind of attribute will be changed ↩
    : group attribute, identity global
attribute, or identity attribute valid only in particular group. If ↩
     the optional 'value' argument is set
then only this value of attribute will be disabled.
Example:
  enableAttribute ig email ann@example.com /group urn:unicore: ↩
      attrType:user:proffession scientist


~~~~~~~~~~~~~~
Command: exit

Exits application

~~~~~~~~~~~~~~
Command: exit

Exits application

~~~~~~~~~~~~~~
Command: getATypes

Lists all known attribute types. Syntax:
  getATypes

~~~~~~~~~~~~~~
Command: getATypes

Lists all known attribute types. Syntax:
  getATypes

~~~~~~~~~~~~~~
Command: getAllEquivalents

Retrieves a list of all identities that are equivalent to the given ↩
     one.
Syntax:
   getAllEquivalents <dn|x509|email> <identity1>
```

```
Example:
   getAllEquivalents email ann@example.com


~~~~~~~~~~~~~~
Command: getAllGroups

Retrieves a list of all groups the given identity is a member of.
Syntax:
   getAllGroups <dn|x509|email> <identity> [implied]
If the last arg is given then also groups implied are returned, i.e ↩
    . if user is a member of group
/A/B then /Awill be returned then too.Example:
   getAllGroups email ann@example.com


~~~~~~~~~~~~~~
Command: getAllIdentities

Retrieves a list of all identities
Syntax:
   getAllIdentities


~~~~~~~~~~~~~~
Command: getApplication

Get an ID of an application submited by a given identity. Prints -1 ↩
     when no application is found.
Warning - note that more than one application may be submited by a  ↩
    single identity.
Syntax:
   getApplication <dn|x509|email> <identity>

~~~~~~~~~~~~~~
Command: getApplicationForms

Lists available application forms.
Syntax:
   getApplicationForms

~~~~~~~~~~~~~~
Command: getApplications

Lists application actions. Arguments filter the resopnse. Negative  ↩
    formId means any formId.
Syntax:
   getApplications [formId] [status]

~~~~~~~~~~~~~~
```

```
Command: getApplications

Lists application actions. Arguments filter the resopnse. Negative  ↩
    formId means any formId.
Syntax:
   getApplications [formId] [status]

~~~~~~~~~~~~~~
Command: getAttribute

Returns value(s) of the attribute of the given element.
Syntax:
   getAttribute <global> <dn|x509|email> <identity> <attribute> [ ↩
       allScopes] [includeImpliedGroups]
   getAttribute <group> <groupPath> <attribute>
   getAttribute <ig> <dn|x509|email> <identity> <groupPath> < ↩
       attribute>
The first argument specifies what kind of attribute will be listed: ↩
     group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  getAttribute ig email ann@example.com /group urn:unicore:attrType ↩
      :user:proffession


~~~~~~~~~~~~~~
Command: getAttributes

Returns all attributes of the given element.
Syntax:
   getAttributes <global> <dn|x509|email> <identity> [allScopes] [ ↩
       includeImpliedGroups]
   getAttributes <group> <groupPath>
   getAttributes <ig> <dn|x509|email> <identity> <groupPath>
The first argument specifies what kind of attribute will be listed: ↩
     group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  getAttributes ig email ann@example.com /group


~~~~~~~~~~~~~~
Command: getAuthz

Lists authorization policy for VO service access
Syntax:
   getAuthz global
   getAuthz group <group>
Example:
  getAuthz group /some/group m
```

```
~~~~~~~~~~~~~~
Command: getContent

Returns the content (subgroups and identities) of the given group.
Syntax:
   getContent <groupPath>
Example:
  getContent /group


~~~~~~~~~~~~~~
Command: getDisabledAttributes

Lists all disabled attributes. Only disabled values are presented.  ↩
    Syntax:
   getDisabledAttributes <global> <dn|x509|email> <identity>
   getDisabledAttributes <group> <groupPath>
   getDisabledAttributes <ig> <dn|x509|email> <identity> <groupPath ↩
       >
The first argument specifies what kind of attribute will be listed: ↩
     group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  getDisabledAttributes ig email ann@example.com /group


~~~~~~~~~~~~~~
Command: getDisabledAttributes

Lists all disabled attributes. Only disabled values are presented.  ↩
    Syntax:
   getDisabledAttributes <global> <dn|x509|email> <identity>
   getDisabledAttributes <group> <groupPath>
   getDisabledAttributes <ig> <dn|x509|email> <identity> <groupPath ↩
       >
The first argument specifies what kind of attribute will be listed: ↩
     group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  getDisabledAttributes ig email ann@example.com /group


~~~~~~~~~~~~~~
Command: getEvents

Displays all events (i.e. contents modification) which occured in  ↩
    the specified period of time. Syntax:
  getEvents [from <yyyy-mm-dd> <hh:mm:ss>] [to <yyyy-mm-dd> <hh:mm: ↩
      ss>]
```

```
If any of range bounds is not specified then it is unlimited.
Example:
  getEvents from 2007-03-28 21:49:00
It will display all events from the specified date till now.

~~~~~~~~~~~~~~
Command: getITypes

Lists all known identity types. Syntax:
  getITypes

~~~~~~~~~~~~~~
Command: getITypes

Lists all known identity types. Syntax:
  getITypes

~~~~~~~~~~~~~~
Command: getMyIds

Lists all identities of the current user
Syntax:
   getMyIds


~~~~~~~~~~~~~~
Command: getNotifications

Lists notifications. Optionally you can query for notifications of  ↩
    a particular action.
Syntax:
   getNotifications [action] [groupFilter]
Example:
   getNotifications addGroup /Math-VO

~~~~~~~~~~~~~~
Command: getNotifications

Lists notifications. Optionally you can query for notifications of  ↩
    a particular action.
Syntax:
   getNotifications [action] [groupFilter]
Example:
   getNotifications addGroup /Math-VO

~~~~~~~~~~~~~~
Command: getPerms

Lists all permissions of specified user
Syntax:
```

```
   getPerms global <dn|x509|email> <identity>
   getPerms group <group> <dn|x509|email> <identity>
Example:
  getPerms group /some/group email ann@example.com

~~~~~~~~~~~~~~
Command: getServerInfo

Retrieves GLUE information exposed by the server.
Syntax:
   getServerInfo [full]
If the optional arg is given then raw Glue 2 XML is printed.  ↩
    Otherwise only a parsed
information (including server's version) is presented.
Example:
   getServerInfo full

~~~~~~~~~~~~~~
Command: help

Provides help. Use without arguments to get generic help or with  ↩
    command name as parameter to get help
on the specified command usage.

~~~~~~~~~~~~~~
Command: helpAll

Provides full help for all commands.

~~~~~~~~~~~~~~
Command: isMember

Checks if the given identity is a member of the given group.
Syntax:
   isMember <dn|x509|email> <identity> <groupPath>
Example:
  isMember email ann@example.com /group


~~~~~~~~~~~~~~
Command: processApplication

Process an application. This method doesn't fulfill any of extra  ↩
    requests attached to the application.
Syntax:
   processApplication <appId> <action> <sendEmail:true|false> [ ↩
       additionalNotes]
Valid actions are REJECT, ACCEPT or REMOVE

~~~~~~~~~~~~~~
```

```
Command: purgeHistory

Deletes all historical content of database which is older then  ↩
    requested. Syntax:
  purgeHistory <yyyy-mm-dd> <hh:mm:ss>

Example:
  purgeHistory 2007-03-28 21:49:00


~~~~~~~~~~~~~~
Command: removeAType

Deletes an existing attribute type, if it is unused (must be also  ↩
    unused in history!)
Syntax:
   removeAType <name>
Example:
   removeAType urn:someTypeTo:Remove

~~~~~~~~~~~~~~
Command: removeApplicationForm

Remove an application form.
Syntax:
   removeApplicationForm <id>

~~~~~~~~~~~~~~
Command: removeAttribute

Removes an attribute. Syntax:
  removeAttribute <global> <dn|x509|email> <identity> < ↩
      AttributeName>
  removeAttribute <group> <groupPath> <AttributeName>
  removeAttribute <ig> <dn|x509|email> <identity> <groupPath>< ↩
      AttributeName>
The first argument specifies what kind of attribute will be removed ↩
    : group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  removeAttribute ig email ann@example.com /group urn:unicore: ↩
      attrType:user:proffession


~~~~~~~~~~~~~~
Command: removeAuthz

Removes authorization policy for VO service access
Syntax:
   removeAuthz <group> a <AttributeName>
```

```
   removeAuthz <group> <o|m>
Use group '/' to modify global policy. Permissions can removed  ↩
    either from the bearer of an attribute or
from the accessed resource owner ('o' case) or from the group  ↩
    members ('m' case')
Example:
  removeAuthz /some/group m

~~~~~~~~~~~~~~
Command: removeFromGroup

Removes an identity from a group. Syntax:
  removeFromGroup <dn|x509|email> <identity> <groupPath>
The first argument is an identity type. Next is value, which is a  ↩
    simple string in case of 'dn' or 'email'
types and a file name with X509 certificate in case of 'x509' type. ↩
     It is the value of added identity.
The last argument is the group path.
Example:
  removeFromGroup email john@example.com /group/subgroup


~~~~~~~~~~~~~~
Command: removeGroup

Removes a group. Syntax:
  removeGroup <groupPath> [true|false]
Separate the parent group path elements with '/'. Optional 2nd  ↩
    argument specifies if the behaviour should
be recursive (defualt is false).
Example:
  removeGroup /g1/subg/groupToRemove true

~~~~~~~~~~~~~~
Command: removeIdentity

Removes a identity. Syntax:
  removeIdentity <dn|x509|email> <value>
The first argument is an identity type. Next is value, which is a  ↩
    simple string in case of 'dn' or 'email'
types and a file name with certificate in case of 'x509' type.
Example:
  removeIdentity email test@example.com


~~~~~~~~~~~~~~
Command: removeNotification

Removes an existing notification.
Syntax:
```

```
   removeNotification <id>
Example:
   removeNotification 3

~~~~~~~~~~~~~~
Command: setApplicationForm

Adds or updates application form. Definition is read form XML file
Syntax:
   setApplicationForm <update:true|false>       <fileWithDefinition ↩
       >
Example:
   setApplicationForm true appForm3.xml

~~~~~~~~~~~~~~
Command: setAttribute

Adds or changes an attribute. Syntax:
  setAttribute <global> <dn|x509|email> <identity> <true|false> < ↩
      attributeTypeAndNameAttributeName> [value1 value2 ...]
  setAttribute <group> <groupPath> <true|false> <AttributeName> [ ↩
      value1 value2 ...]
  setAttribute <ig> <dn|x509|email> <identity> <groupPath> <true| ↩
      false> <AttributeName> [value1 value2 ...]
The first argument specifies what kind of attribute will be changed ↩
    : group attribute, identity global
attribute, or identity attribute valid only in particular group.  ↩
    The boolean argument specifies if existing
attribute with the same name should be updated.
Example:
  setAttribute ig email ann@example.com /group true urn:unicore: ↩
      attrType:user:proffession scientist


~~~~~~~~~~~~~~
Command: setAuthz

Modifies authorization policy for VO service access
Syntax:
   setAuthz <group> a <perm> <AttributeName> [value]]
   setAuthz <group> <o|m> <perm>
Use group '/' to modify global policy. Permissions can assigned  ↩
    either to the bearer of attribute or to
the accessed resource owner ('o' case) or to the group members ('m' ↩
     case')
In any case permissions are specified as string with syntax:
   <r|-><f|-><i|-><w|->
where 'r' is read permission, 'f' is full read permission, 'i' is  ↩
    identityCtl permission and 'w' is write
permission.
```

```
Example:
  setAuthz /some/group m -f---
It will assign members of /some/group (and it's subgroups) the  ↩
    fullRead permission and remove all other
permissions (if were set).

~~~~~~~~~~~~~~
Command: setIdentityStatus

Changes the given identity status to disable or enable it.
Syntax:
   setIdentityStatus <dn|x509|email> <identity> <true|false>
The last argument equal to 'true' will make the identity enabled ( ↩
    active) and 'false' will disable it.
Example:
   setIdentityStatus email ann@example.com false


~~~~~~~~~~~~~~
Command: setTime

Set time in the past for the subsequent queries. After setting the  ↩
    time all *query* operations will query
the past, historical contents of the service.
Useful only in interactive mode
Syntax:
  setTime [yyyy-mm-dd hh:mm:ss]
Without the arguments it will return to normal operation on current ↩
     contents. Example:
  setTime 2007-01-01 23:56:00

~~~~~~~~~~~~~~
Command: submitApplication

Submits a new application. It is read form XML file.
Syntax:
   submitApplication <fileWithDefinition>
Example:
   setApplicationForm app1.xml

~~~~~~~~~~~~~~
Command: updateAType

Adds or updates existing attribute type
Syntax:
   updateAType <name> [shortDescription] [longDescription]
Example:
   updateAType urn:newType 'Dummy type' 'Dummy type used to express ↩
        foo with bar values'
```

```
~~~~~~~~~~~~~~
Command: updateCSRApplication

Updates an application containing CSR only, when CSR is processed ( ↩
    i.e. accpted or rejected) by a CA.
Syntax:
   updateCSRApplication <fileWithCSR> <accepted:true|false> < ↩
       sendEmail:true|false> [<fileWithSignedCert>]
File with signed certificate is needed if accepted is true.
```